# HIPPI-6400 INFORMATIVE ANNEX SECTION DRAFT

Revision 0.1

**DRAFT FOR HIPPI-6400 WORKING GROUP MEETING 5/9-10, 1996**

**This draft addresses the topic of fair bandwidth allocation and the HIPPI-6400 VCs, and could be subtitled "How to Get the Behavior You Want From Your HIPPI-6400 Connections". It attempts to deal with differences which users and and systems designers will see between HIPPI-6400 behavior and the predecessor HIPPI channels, and will eventually suggest mechanisms which can be used to ensure fair bandwidth allocation and predictable behavior.**

This **very preliminary** first draft is provided to stimulate discussion and in in response to an action item assigned to Marti Bancroft at the March HIPPI-6400 working group meeting. In addition to comments from those attending the March meeting, reflector postings from Philip Cameron, Francois Gaullier, John Mullaney, Michael McGowen, Dave Parry, Steven Poole, John Renwick, Roger Ronald, Rami El Sebeiti,Burton Smith, Don Tolmie, Thomas Torrico, and Tim Voss wereedited and incorporated into this draft.

**There are seven major sections:**

- **1.0: Review of Definitions and Differences - HIPPI-6400 Compared With HIPPI-800 and HIPPI-1600**
- **2.0:Use of VCs for Different Types of Traffic**
- **3.0: The Meaning of a "Connection" In HIPPI-6400**
- **4.0: Review of Physical Bandwidth Allocation and Some Usage Scenarios (eventually this would be the section in which modelling and simlation data could be reported)**
- **5.0: Suggested Strategies (for Hosts and Adapters) To Achieve Whatever Performance and Behavior Under Load is Desired**
- **6.0: Comments on Switches (to be added)**

- **Comments on SNMP for Mixed HIPPI-6400/1600/800 Networks (to be added)**

***The following paragraphs are intended as background for discussions during the evolution of the draft. They are not necessarily what would go into a final standard annex but rather are provided to stimulate discussion.***

It is anticipated that the approach defined in the HIPPI-6400-PH standard for the use of VC0 and VC3 to accomplish bulk data transport over the HIPPI-6400 media would be further defined in HIPPI-MP for use with multiple HIPPI-800 (or HIPPI-1600) connections over HIPPI-6400.

However, in neither discussion has the draft standard component defined specific mechanisms for assuring “fair bandwidth sharing” among the four VCs defined in HIPPI-6400, or among the multiple legacy channels which may be multiplexed over a single VC3 channel. There are congestion control features proposed for the HIPPI-6400 fabric (referenced in section 1.0 and detailed in HIPPI-6400-PH). However, while these may be very effective for guaranteeing that a form of deadlock does not occur, they provide a probablistic performance as opposed to guaranteed level of service. For some applications, any level of service (other than none) may be acceptable. For others, the traditional HIPPI connections were used because they provided high-bandwidth bulk data transfer with extremely predictable (i.e. deterministic) behavior.

What is defined as “fair” is definitely in the view of the system architect and could range from oversubscribing the HIPPI-6400 channel and assuming that “fair” means everyone gets a chance at service, to dedicating the channel to a single device doing bulk data transfer. For HIPPI-6400 to be a success as an advancement for the legacy HIPPI channels, we need to provide a mechanism to accomplish either identical or improved behavior as compared with that achievable using legacy channels and software.

“Fair bandwidth sharing” is expected to occur on VC3 in part due to “chunking” and HIPPI-MP. By partitioning larger transfers into smaller units for transmission, multiple transfers originating on slower media (for example, six simultaneous HIPPI-800 transfers) can appear to the host as simultaneous 800 Mbit/s transfers delivered in periodic 6400 Mbit/s bursts, assuming of course that the “translator” (refer to HIPPI-MP draft) places each segment on the HIPPI-6400 media in round-robin order. Similarly the host providing multiple HIPPI-800 streams to a HIPPI-6400 interface would be expected to provide segments (or entire blocks) for each stream in a round robin fashion to the adapter, to sustain each at maximum rate.

At some point during the standards process, the HNF will need to consider how HIPPI-6400 fits with the predecessor standards in two areas:

- interoperability (ideally legacy systems would interoperate without hardware or software change with HIPPI-6400 systems); and

- user expectations about behavior based on experience with the legacy standards, and how to insure that users (system architects) have the information and tools to be able to ensure the behavior they want when HIPPI-6400 is incorporated into their systems

This draft attempts to provide some words addressing the later, and a placeholder into which definitive modelling and test data could be later inserted.

**Definitions in this draft:** The term “user” refers to an application, or an individual process within an application. The time-distribution of data flow for a user is referred to as a “stream”. A “stream” is a set of blocks or packets (which may themselves be further broken up into chunks as part of the transfer). High performance is defined as being 800 Mbits/s or greater.

# 1.0 Review of Definitions and Differences - HIPPI-6400 Compared With HIPPI-800/1600

## 1.1 Brief Summary of HIPPI-800/1600 Behavior

From a systems perspective, the legacy HIPPI standard could be considered a single channel. No data could be transmitted without a “connection”. Only a single user could have a connection at a time, and having a connection guaranteed exclusive use of the entire bandwidth of the channel for the duration of time the connection was held. Releasing the connection between blocks or packets was used to allow the possibility of sharing the channel between different users. Complexity was kept to a minimum.

This connection-oriented behavior was both feature and a liability.

**A Major Strength of Legacy HIPPI:** It was a feature for users who wanted to use HIPPI-800/1600 as a dedicated but switchable channel (i.e. for peripherals or special devices), as holding the connection guaranteedownership of the entire channel and path - even through a complex of switches. This made the performance very deterministic. The sustaineable bandwidth was determined by:

- the protocols being used
- transfer sizes
- block (or packet) sizes.

For sufficiently large blocks and transfers, using light-weight protocols (HIPPI-FP) and performing transfers directly to/from user space to enable larger block sizes and to bypass system buffers, a sustained rate close to the underlaying physical bandwidth was possible and has been demonstrated by at least one vendor (editor’s note: I have personally run such using Cray PVP HIPPI-800 and 1600 channels; I wanted’ to say “several vendors” here but didn’t have the data. Those of you who have run such tests and achieved >95 MB/s or >190 MB/s please let me know which systems so we can switch this to “several computer manufacturers”. Thanks!) The dedicated channel could, however, be switched. This behavior was desirable for disk and tape devices (in which bulk file transfer was used) and for certain special devices. The overhead to make and break a connection (even including host software times) was usually a microscopic portion of the total transfer time.

**Legacy HIPPI Wasn’t Ideal for IP Traffic:** Always requiring connections was a liability for typical IP network traffic. It was (most of the time) undesireable to dedicate a HIPPI channel to a single IP user, so shared IP traffic would usually occur via OS services. First, the multiple users had to contend with making/breaking connections for each piece of their transactions. The packets used were typically much shorter than those used for bulk data transfer, so the connection process overhead was more significant. On many systems, the best case performance of an individual IP stream was a very modest fraction of the innate channel capability, and multiplexing IP streams did not necessarily add linearly (or at all!) to the total sustained throughput. Combiing bulk data transport traffic and IP users on a signle HIPPI channel was extremely unwise. Even if bulk data transfer users were “well behaved” (dropped and re-established connections for each block), mixing traffic types would often result in erratic behavior for both types of users (sometimes widely fluctating performance at the applications level, and (in some systems) many timeouts for the IP traffic when very large blocks of bulk transfer data were transferred to slower devices. Some vendors therefore strongly recommended that users not mix traffic types on HIPPI-800/1600 channels. HIPPI-1600 is not widely used and the predominant applications for it have been high-speed bulk data transport, not IP.

One of the goals of the discussions leading to the HIPPI-6400 project was to emulate the HIPPI-800/1600 example by providing a **simple** specification (therefore **implementable**) with **no options** (to maximize interoperability probability). Another suggested goal was testability. It was easy to test whether HIPPI-800/1600 hardware was working - connecting the input and output cables on a single host allowed checkout without any HIPPI-based devices. At least one vendor found this a convenient way to do maximum sustainable I/O testing.

One reviewer pointed out that a very nice side effect of having to write clear definitions of "stimuli and responses" (to ensure testability) is that this would be a natural governor on options and complexity creeping into the HIPPI-6400 draft standard. Some reviewers believe that HIPPI-800/1600’s simplicity and efficiency are why it is the leading bulk data transfer high performance standard.

## 1.2 Summary of HIPPI-6400 VCs and Possible/Expected Behavior

In contrast, a HIPPI-6400 physical channel is actually inherently multiple logical channels. (Editorial comment: this isn’t say say it is a liability - and for general networking use it is **definitely** a strength - but it is a MAJOR difference to legacy users.) The background below and discussion of approaches to VC interaction is provided to illustrate at least some of the implications of this from a user and system architect perspective. The editor believes this is an area in which modelling is highly desireable before the approach is finalized.

**Background :** There are 4 virtual channels (VCs) within each HIPPI-6400 physical link (really 8 VCs if one includes the full-duplex nature of HIPPI-6400). At the February 1996 HIPPI meeting the VCs were assigned to traffic types as:

- VC0 = Connectionless, up to 2 KByte messages (used primarily for control)
- VC1 = Connectionless, up to 128 KByte messages
- VC2 = Connectionless, up to 128 KByte messages
- VC3 = Connection-oriented, connection setup done over VC0 (unlimited message size) (now called scheduled messages, and described in detail in section 7 of draft rev0.2 of HIPPI-6400-PH)

Three approaches were considered for how these different VCs would interact on a shared physical channel, i.e., from which VC is the next 32-byte micro-packet transmitted? A message takes a whole VC for the complete time to transfer the message, i.e., one cannot multiplex multiple message parts on a VC.Whatever is used, it must be relatively easy to implement as this part is being done in silicon.

The three approaches were:

1. A round-robin which switches between VCs every 256 micro-packets,or until the active VC runs dry. (256 micro-packets = 8 KBytes or 10.24 usec) For example, if VC1 is transferring at full blast,then after sending 256 VC1 micro-packets the link would scan VC2, VC3, and VC0, in that order, for micro-packets to send. Now if VC3 and VC0 both have micro-packets to send, and VC2 has nothing,then up to 256 VC3 micro-packets will be sent before starting theVC0 micro-packets.
2. Immediate priority for VC0; other VCs on a round-robin as above. For example, when a VC0 micro-packet is available, send it immediately independent of which VC was transferring, and what micro-packet number it was on.
3. Messages are kept flowing once started, so that if a VC continues to have micropackets to transmit and has credit available, that VC would have priority over other VCs.

Draft Rev0.2 of HIPPI-6400-PH has (section 8.4) either #1 or #3 as the mechanisms. This editor suspects the topic needs more discussion so has included it here.

Some of the implications are (comments from several reviewers incorporated):

**Under approach 1: all round-robin .**

A connection request over VC0 may have a one-way latency of up to 30.72 usec if all of the other VCs are busy, and a total round-trip latency of up to 61.44 usec. The latency would be less than 1 usec if the other VCs were idle. The long latency may not be a concern when you consider that the destination probably cannot make a connection immediately anyway (i.e., it is already using VC3 to transfer to some other Source).

HIPPI-800 switches currently exhibit less than 1 usec circuit setup latency, and approach 1 makes this aspect of HIPPI-6400 slower than HIPPI-800. Is this acceptable?

The traffic is allocated fairly between the VCs; no one VC can hog the bandwidth. Approach 1 seems likely to work very well when the majority of the traffic is large block transfers over VC3, since the connection overhead (VC0 latency) would be amortized over a large block. One exception - although only onemessage would be occuring on VC3 at a time, the effective bandwidth seen could vary depending on activity on the other VCs.

For other uses of VC0, e.g., sending RPCs, out-of-band control attempts (the new ADMIN packet type?), or cache line updates, a 30 usec latency may be highly undesirable.

**Under approach 2: VC0 has priority.**

Short control messages can cut through with minimal latency whether or not there is traffic on the other VCs.

Concern has been expressed that people can mis-use VC0, using it for all traffic and hence degrading its performance to worse than the worst case seen with approach 1. However, since VC0 is used for control messages, its primary user is likely to be the OS and perhaps this concern will be mitigated.

Approach 2 seems better for machines using lots of short high-priority messages and some long messages. I.e., the short control messages won't get blocked behind the long messages for as long.

The latency on VC0 seems more deterministic.

Lots of traffic on VC0 can essentially starve the other VCs

**Under Approach 3: Messages with continuing data keep going.**

This would seem likely to work well in a lightly loaded system but might produce very erratic performance under heavy and especially under variably heavy loading.

**Further Discussion:**

**What is desireable is definitely affected by intended use of HIPPI-6400,as the following reviewer comments illustrate:**

**Determinism:**

It is key to have a mechanism that is deterministic. One needs to be able to predict precisely what the performance is going to be (at least on one channel). Performance includes both sustained bandwidth and latency.

The current draft specification of HIPPI-6400 is that Virtual Circuit (VC) messages are not interleaved within the VC. Once a message starts along a path between source and destination, it must complete prior to the use of that same

VC for any other message. For bulk data transfers on VC3 this means that they will be able to function essentially as they do on legacy HIPPI (i.e. appear to have the entire channel bandwidth for the duration of the message) UNLESS:

- VC1 and VC2 traffic is occuring, or
- VC0 traffic is for more than just messages associated with controlling the bulk data transfer in process, or
- the destination or host can't sustain the rate (run out of buffers, etc), or
- an error occurs

**A Systems Design Issue:**

Since the physical definition of the channel does not prevent VC0, VC1, or VC2 activities from occuring simultaneously with VC3 bulk data transport, and vendors will need to supply an implementation which allows for all VCs to be used as provided in HIPPI-6400-PH in order to be compliant (correct, right?), system and/or user software will probably need to be configurable to allow assurance of determinism for VC3. This can be either very simple or extremely complex. The simple option could have just two modes:

1. If the configuration is for "networking" (mode name tbr), the multiple diverse users sharing the HIPPI-6400 channel and using separate VCs will receive a probablistic level of service (i.e. the latencies and bandwidth they experience will vary both as a function of the total loading and as a statistical process of message sizes, arrival times, etc. Modelling a variety of scenarios is recommended to provide guidelines for system architects. Switch behavior will add another set of variables, since three of the four VCs are "connectionless". The host software would not be expected to restrict loading in any way, other than by number of buffers configured. Initial guesstimates (anybody have modelling data yet?) would be that this would provide higher overall throughput for IP-type traffic (as a fraction of channel bandwidth) than legacy HIPPI, due to reduced overhead - IF the host made networking software suitably efficient.
2. If the configuration is for "bulk data transfer", the mode would be intended to provide a HIPPI-6400 equivalent of a held connection. To accomplish this, traffic on VC1 and VC2 must be stopped during the bulk data transfer, and the only VC0 traffic which could be permitted would be that related to the bulk data message. This has the effect of dedicating the channel to a data stream. Obvious suggestions are to transfer data to/from user space (i.e. don't use OS buffers) for efficiency and to allow very large blocks, and allow this mode to be set only by priviledged programs.

Several of what may be an infinite series of more complex options could be (in addition to the two above):

3. Have some scheme for reservation of a guaranteed level of service, either bandwidth or latency, or both, for the VC3 traffic. Host software would then allow only as much non-VC3 related traffic as would be less than the remaining bandwidth (how much less to provide the guaranteed level is tbr - more modelling again). Although this sounds complex, in the (very probably) event

that the recipient of the VC3 message could not sustain the full HIPPI-6400 bandwidth, this would improve overall utilization of the channel resource.

4. Using mode 2, use the proposed HIPPI-MP to mux the VC3 traffic (assuming suitable “translator” between HIPPI-6400 and multiple legacy HIPPI channels)
5. Use mode 1 plus HIPPI-MP to improve utilization of VC3.
6. Have some scheme for a guaranteed level of service for each VC, not just VC3.
7. etc etc

**Those who need to fully understand and predict in advance and to ensure specific and deterministic system behavioral characteristics using HIPPI-6400, will have to perform more analysis and system configuration work (plus possibly some modifications to current host HIPPI support) will likely be required than when using legacy HIPPI.**

**Latency and Packet Interleaving:**

Being able to lower the latency to the minimum is also very important to many reviewers. Latency is considered a key parameter, especially for short messages.

Some reviewers believed the channel switched too slowly (every 256 micro-packets) between VCs?If the hardware could switch faster, it should, because that would latency. If hardware were fast enough to interleave active VCs on a per-micro-packet basis, latency would be at a minimum. Could this be left up to the implementation? If so, what about interoperability?

Packet by packet interleaving guarantees that the latency for four messages can be the absolute worst case. When you require all four VCs to share on a single micro-packet basis all of the messages are delivered in approximately the message length times 4. In other words, this approach treats everyone equally badly. It means in a busy system that no VC delivers messages at a rate faster than 200 MB/s. (Those wanting bulk data transfer at speeds faster than HIPPI-1600 are likely to reject this approach.)

The problem with fine-grain interleaving is that it penalizes all traffic when multiple VCs have similar size messages. If VC 1, 2, and 3 are all sending 128KB messages across the same physical link, the last to get through will suffer a latency of 480usec. If the VC's are interleaved at the micropacket level, then *all three* of the messages will suffer this latency assuming that the accumulated end-to-end CRC must be checked before the data is used. If the messages were interleaved on a per-message basis, the three messages would suffer latencies of 160, 320, and 480usec based on the order in which they got queued. This solution has a lower average latency. However, a small or moderate size message could get queued behind a number of large messages and have poor latency. Don Tolmie’s original proposal attempted to form a compromise by choosing an interleave granularity (8KB) large enough to assure reasonable latency when multiple VC's were transferring moderate size messages.

As he pointed out, it might also be desireable to have VC0 get priority in order to provide a lowest-possible latency path for high priority control messages.

Interleaving on a per-micropacket basis is certainly the simplest to implement if higher average latency is acceptable and the cost of hardware is not greatly increased.

Leaving interleaving and prioritization up to the implementation is not likely to be acceptable given that the priority and interleave scheme will have a substantial effect on the overall behavior of a switch. Also, system architects would have to ascertain and analyze each scheme used by a potential vendor to ensure that it would both give desireable behavior and interoperate with other equipment already present or proposed for the system. This seems counter to the purpose of having a standard.

**Implementation Ease:**
Approach 2 would be the most straight forward from an implementation viewpoint. One would provide VC0 u;timate priority but only on 256 burst boundaries and do round-robin for the other VCs.This would provide reasonable determinism and a shorter maximum latency. The priority logic is only a slight variation from a simple round-robin and there is no exception state (last one wasn't a full 256, odd vs. even count etc...)information to save.

**Discussion of the Need for Priority on VC0:**

If latency is minimal, there needn't be any priority as in approach 2. All VCs can be at the same priority, serviced round-robin. This is fair, since they're all sharing the full bandwidth of the link.

In a typical world, any of the suggested approaches probably provide good response and reasonable fairness. These maximum latency numbers only apply when the system is busy on at least a couple of VCs. Leaving it to the vendor implementations may be appropriate.

But - what about the need to get a critical control message through on VC0? Examples might be to stop a runaway host, send a reconfiguration messages,...is “immediate” required, or would VC0 simply have to be the next to go when the currently-transferring 256 micropacket transfer completed? “Immediate” implies retransmission of an interrupted 256 micropacket transfer - would this be handled in the silicon?

**Message Size Limits (Reference section 6.2, HIPPI-6400-PH Rev0.2):**

One reviewer feels the limit on message size for VC1 and VC2 should be 64K bytes, not 128K, since the application envisioned is network protocols (e.g. IPv4 and IPv6). MTUs near 64K are being chosen so that there is room for additional hardware headers within a 64K buffer (i.e. message). For instance, legacy HIPPI's 65280 allows for 256 bytes of hardware headers, more than enough. IPv6 has a "jumbogram" option for packets greater than 64K. It requires an optional header, and is expected to be used only between locally connected supercomputers. We should consider requiring that these go on VC3 as bulk data transfer.

**Implications for Switches:**

Round robin is preferable, with no priorities on any VC. We should definitely allow the ability to switch on every micro-packet. If sources find it more effective to do long groups of sequential micro packets then that is still possible, but it doesn't require a limiting definition on the switch's operation.

HIPPI-6400 switches under these constraints are considerably more deterministic than current HIPPI-800 switches because of the undefined arbitration response to Camp on requests in HIPPI-800. Do we need to define what the characteristics of arbitration are between ports on a HIPPI-6400 switch? This is undesireable if it gets too complex, but this has considerable effect upon deterministic behavior when switches are involved.

For switches, a time window mechanism could be considered, that is, when multiple requests arrive simultaneously (or within a fixed window of time for the same destination, they are all satisfied first before any new requests for that destination are allowed into the arbitration. The size of the window is tbr. A good first pass to the window size might be a micro-packets time period (40ns). Consider aas an example a 32 port switch. Assume that all 32 sources are sending single micro-packet messages to the same destination on all 4 VCs. The worse case latency for any one micro packet is 32 times 4 times 40ns equals 5.12us plus a taste more for buffer effects - not bad. If VC0 is limited to 2K then the worse case latency for a VC0 message when all 32 ports are sending on all 4 VCs equally to a single destination and all are sending maximum VCO sized messages is: 327.68us.More meaningful is when VC1, 2 and 3 are busy to a destination from 3 separate sources respectively then the latency for a single maximum sized VC0 message is only 10.24us.

### 1.2.1 Fabric Congestion Control in HIPPI-6400

One charactersitic of HIPPI-6400 is that Virtual Circuit (VC) messages are not interleaved within the VC. Once a message starts along a path between source and destination, it must complete prior to the use of that same VC for any other message.

Bad behavior on the part of a destination or a source could potentially clog things up for both the destination and source. This includes running slowly or stopping (probably due to errors). The symptoms are likely to be frustrating for a network administrator - works OK when running diagnostics (assuming no hard physical errors), but under load results are either highly variable or outright hangs.

The first fix for preventing clogs is to require that sources and destinations should be able to handle most traffic at full rate. Overruning conditions should be avoided, perhaps even by dropping data when the host cannot keep up (IP comes to mind). In practice, this may not be enforceable at first, since many next generations systems do not have I/O channels which could absorb HIPPI-6400 and this wouldn't apply unless they were using legacy HIPPI and a translator. In that case, this could be a requirement levied on the (new) translator products.

Even if all of the above case are dealt with, system errors may create clogs when they occur in the middle of a transfer. The standard should (hopefully) address steps link ends need to take to assure that clogs do not persist indefinately. The portion of the standard dealing with switch behavior should call out steps switches need to take to assure that clogs do not persist indefinately or propagate through cascading switches.

There are four fundamental things in proposed HIPPI-6400 which to help resolve these situations:

1. Transactions are all closed-loop at the endpoints. This allows the endpoints to have timeout mechanisms, and to act appropriately when they are tripped.
2. Fabric elements implement a tail timeout mechanism. Tail timeouts occur when a message has been started and the VC buffers see no activity for a very long time.
3. Fabric elements implement a deadlock timeout mechanism. Deadlock timeouts occur when a micropacket sits in the VC buffers for a very long time and is unable to be forwarded to the next fabric element (due to a lack of VC credits).
4. Fabric elements implement a timeout for config transactions which are unable to be forwarded to the next fabric element. (This is necessary because there is a request/response resource conflict in the config block, which can cause pairs of config blocks to mutually deadlock.)

## 2.0 Use of VCs for Different Types of Traffic

### 2.1 Chunking for VC3

Proposed HIPPI-MP deals with multipexing legacy HIPPI channels on a HIPPI-6400 channel using chunking. HIPPI-MP defines how an upper-layer protocol would split up a large file and simultaneously send it over several HIPPI-800 interfaces to get an increased transfer rate. This requires a new ULP-id in HIPPI-FP, and an amendment to, or a revision of, HIPPI-FP. Discussion during HIPPI-MP review is included here because it sheds light on some performance implications of chunking and performance. This area would also benefit greatly from performance modelling.

The HIPPI-MP proposal breaks the VC3 data into "chunks", with each chunk being 2 KBytes of data and a small header. Each chunk is a separate message. We should also consider making the chunk size variable up to some large maximum size. In thinking about implementation, there are two places that need to be able to handle chunks. One is in a translator (probably a hardware beast) and one is in any host that wants to be able to communicate in this manner.A hardware based translator would be required for a 2K chunk. However, there are several reviewers who believe a 2K chunk is too small to be practical.

To clarify the behavior with "chunking", envision a host over HIPPI-6400 trying to stripe to 8 HIPPI-800s at full rate.

The packets are: Per HIPPI-800 = just under 50,000 2K packets per second to achieve full rate of 100 MByte PLUS an equivalent number of command packets PLUS an equivalent number of response packets times 8 HIPPI-800 channels possible per HIPPI-6400 channel.

This results in a rough number of 1.2 million packets per second to handle on the host end. For most current hosts, this is an unsustainable transaction rate.This is for only one HIPPI-6400 channel, and the host may need to support more than one.Encouraging applications to run chunk sizes that are more in the 1 MByte range seems more achievable.

For legacy HIPPI, one reviewer encouraged a fixed chunk size of 16K bytes and the use of class 2b or 4b packets exclusively. Using a smaller granularity would entail more work in segmentation and reassembly, especially for non-shared memory systems such as networks of SMP's, and was regarded as less flexible with respect to timing and network configuration. Using legacy HIPPI as an example, striping with a chunk size of 16K bytes means 16 (resp. 8) D2 bursts would be needed for 32 bit (resp. 64 bit) HIPPI. This would keep the D1 overhead reasonable. The overhead due to fragmentation for transfer lengths not nearly divisible by 16K*stripe_width will be the time to transmit 16K bytes, which is approximately 160 microseconds for 32-bit HIPPI and 80 microseconds for 64-bit HIPPI. These times are short compared to the time to initiate a transfer on most systems, so the overhead from this source will also be reasonable for legacy HIPPI. (Depending upon the actual host overhead observed for HIPPI-6400, this may not be the case since one would expect hosts supporting the full HIPPI-6400 rate to have more efficient mechanisms.) A fixed stripe chunk size would simplify the MP layer. If only class 2b or 4b packets are used, it is relatively easy to manage data reassembly in a shared memory system, and there is room for several layers of protocol in the D1 area. If you allow any old packet format it is harder to layer protocols flexibly; the data will begin after the last byte of protocol stuff, whereever that is, so the protocols are harder to make composable. Consequently packet format is expected to be well-defined for HIPPI-6400.

In an effort to clean up some things in HIPPI-FP to make it more efficient, the D2_Offset was identified as a likely candidate for deletion. Suggestions for other changes, additions, deletions, etc.to HIPPI-FPwere also requested.

However, HP's legacy driver uses a D2_Offset other than zero. This is for the level 2 interface which provides LLC 802.2 services. Using this interface, a user data packet is encapsulated:

- by the LLC header (3bytes): DSAP (1byte), SSAP (1byte), Control (1byte).

- LE header (24 bytes),

- FP header (8 bytes).

It was very convenient and more efficient for the transfer on the board, to have the data on 4 bytes boundary. And we use 5 bytes D2_Offset to complete the LLC header.

HIPPI could take advantage of the D2_Offset, each time there are curious size headers.

One reviewer suggested adding a parameter to the chunk header list included in the original proposal to move connection "chunks" across VC3.This proposed parameter would typically be used when large (>1 MB) packets moved from HIPPI-6400 host to HIPPI-6400 host.

The originally-proposed chunk header (reference current draft of HIPPI-MP) would contain such things as:

- Source-id
- Message-id
- Chunk sequence number
- Address offset
- Data_size in the chunk
- Last_chunk indicator

The parameter to add is "I-Field". Adding a path specifier to the list would allow an alternative technique that can substitute for the alternate pathing capability common in HIPPI switches. When the sending system moves to establish the connection across the fabric, it is not in a good position to choose one of multiple paths to the destination (assuming they exist). But the destination system is in a good position to realize the state (busy or not) of its input ports (and perhaps even future scheduling for VC3).

Consequently, the destination system can specify the preferred port for the source to use by passing an I-Field. This allows the source to send its data down the next free path (or a path selected for any alternative reason that the source would desire and implement).

## 2.2 Sharing HIPPI and Flow Control, or “To Drop or Not To Drop? That is the question...”

The editor has included reviewer comments here because what is reasonable really depends on whether one is using HIPPI-6400 as a peripheral channel or as a network medium.

For 2 applications using HIPPI-FP on the same system, with packets received for each of these applications. What happens if for any reason, one application is not ready to receive its packets? (a flow control at application level for example) Does it flow control at HIPPI-PH level, preventing the other applications from receiving their packets? Or (as the memory resources are limited), should the application be obliged to drop the incoming packets for that application?

It is suggested that each application needs to be careful to keep read requests active all the time, so there is always a buffer available to receive an incoming HIPPI message.

The following code for an application at receive side does not use multiple buffering:

```
while (1)
{
 /* receive a packet */
 read(buffer);
 /* process the packet */
 process(buffer);
}
```

When executing process() routine, there is not a read active. And there is always a case, when the read side is as quick as the write side. And that's why legacy HIPPI provided the flow control at physical layer (READY/RESUME). If an application can't keep a read request posted or gets behind, then the driver should discard its input message. In order to keep a read outstanding while you're processing, one has to use a non-blocking read call such as receive(). Some operating systems have non-blocking read() and write() functions.

Another way is to do a request/response sort of protocol, where you don't get a new message until you've processed the last one. Then at least there can be a very short window between sending the response and reading the next request. Of course, you can lose the CPU in that time, which breaks the application. Also this approach may not give you the throughput you want.

One reviewer notes that it can be difficult to make applications run this way. It's much easier if the operating system can buffer messages for the application, which is why sockets tend to work a lot better than raw memory-to-memory applications in their opinion.But - what happens if no more buffers are available at operating system level?

If there is a limited number of buffers configured for the HIPPI subsystem this can happen. And even with very big number of buffers, at 100MBytes/s (let alone 800 MB/s for HIPPI-6400) system buffers can quickly be consumed. If the operating system buffers packets for the user, there is always the possibility that it may run out of buffers and have to drop a packet,and then the user needs a protocol that can detect this and resend the packet. From the point of view of sharing a legacy HIPPI channel,one reviewer believes the host should send the READYs even if it is out of the buffer space.In other words, the host must quickly discard packets when there is no buffer, and leave it up to the application to recover. (The analogous mechanism in HIPPI-6400 would be ensuring available credits.)

This reviewer believes these problems lead inevitably to TCP or an equivalent reliable transport. Users may decide they'd rather have reliability with reduced throughput than a raw-mode solution that doesn't work all the time.

However, some customers have successful bulk transfer implementations using transfers direct to user buffers. This requires allocation of sufficient memory buffers to ensure one is always available to keep the data stream going at sustained rate. Customers doing this are more interested in high performance and predictable sustained rates rather than reliable multi-user use of the channel with the users being unaware of channel state or behavior.

Whatever is the recommended method (flow control or drop the packets), it should be clearly specified for the standard HIPPI API.

Another reviewer noted that,from the API side one should not have to worry about flow control unless the application specifically desired to do so. Otherwise, from tha Application Programmers point of view, a reliable connection is needed to deliver data on request reliably (but without a guaranteed bandwidth?) If the user wants to use OS Bypass or something like that (THIPPI, user space ansychronous I/O) then the userknows a priori that there are some defined difficulties. This reviewer has used FrameBuffers on IBM,CRI,SGI and HP and the CM-200 and the CM-5 and never seen the problem you are talking about, at the APPLICATION level. The whole idea of an API is so that the user does not have to deal with flow control.

HOWEVER, frame buffer applications are easy--they're output only. The problems above occur when you're trying to read the HIPPI, or rather, when the driver receives a HIPPI packet for which there is no read, i.e. there is no buffer to put it in. I agree with Steve that the application shouldn't have to worry about flow control. However the application WON'T get a reliable connection with raw access to a shared HIPPI. In order tohave a reliable connection with a raw interface to HIPPI (not TCP/IP), the application has to own the channel and the driver must not send READYs unless the application has supplied a receive buffer. This works for legacy HIPPI - how can it be assued for HIPPI-6400?

An assumption of the proposed problem is that the data is processed faster than it is read. If this is not the case the system would have to buffer more and more data as time goes on since processing can't keep up. This would result (ultimately) in running out of memory. In the above case, the rate of arrival of packets has to be carefully managed to permit the processing to complete before the next packet starts to arrive.

One reviewer thinks that the arriving packet should not be discarded and the HIPPIconnection should flow control if a read is not outstanding. It is easy to imagine back to back reads that happen faster than the system can return from a read and start the next read. The operating system can (usually) buffer a very limited amount of user data. At HIPPI data rates (especially HIPPI-6400) it may be best to assume that system buffering is not really available.

Therefore HIPPI should flow control - hence the credit mechanism proposed for HIPPI-6400.

In some host operating systems non blocking IO permits the read(or write) to complete immediately when there is no available data (or no room to queue data) rather than to wait. Since the read (write) completes the buffer is available to the application for other use (Unix semantic).

On a system that supports multi-threaded processes, some overlap of read and processing could be acheived if the read can be managed by one thread and the processing can be performed on another thread. Buffers can be passed between the receiver and the processor over a FIFO. Processed buffers can be returned to the reader using a FIFO. Again, the relative flow rates and number of buffers need to be calculated such that there is always an available read buffer.

Another way is to do a request/response sort of protocol, where you don't get a new message until you've processed the last one. Then at least there can be a very short window between sending the response and reading the next request. Of course, you can lose the CPU in that time, which interrupts the application. This approach isn't likely to give maximum throughput.

This reviewer did not like dropping packets. Most systems tolerate a very low data loss rate. The typical communication subsystem expects almost all packets to arrive properly. The "rare" lost datagram can be detected and recovered. When the loss rate gets higher the system tends to spend more and more time in the recovery process which can result in more dropped datagrams - at worst case resulting in the channel being used for primarily retries with not real data transfer occuring. HIPPI-6400 should attempt to deliver all non-corrupted data and use recovery mechanisms to recover the occasional lost packet.

This becomes more challenging when sharing the channel between two ormore applications. When not sharing, one can flow-control to make sure messages don't get lost. But with a shared channel, the legacy HIPPI driver has a problem--as long as one application has a read up, it's got to issue READYs so it can read the next message to find out who to deliver it to. (For HIPPI-6400, credits for that VC must be available.) If the message is for an application that isn't reading, then you have to penalize that application so that the other one, that IS expecting something, can receive it. It's just a trade-off between reliability and bandwidth. If you put the burden of reliability on the application, then at least you can deliver some bandwidth.

From a system view, the problem can be expanded to "N" processes sharing a HIPPI channel. Of the "N" processes "M" will not have a read outstanding at any given point in time. Given that the arriving packet can be long, the system can run out of buffer space before the entire message has been received. What then? What happens when the target process posts a read while we are in the process of buffering the data? What do we do when "M" is almost as large as "N" (statistically possible) ...and one of the "M" processes is being flooded with packets? The time taken to buffer the message impacts all user processes in the host. The reduced memory (due to buffering) impacts paging for the whole system. The host system and all processes suffer performance loss.

This reviewer further noted that reliability is very difficult for applications to implement and many mechanisms are fairly costly in time when recovery is needed. He therefore feels strongly that the system should never drop messages. It is bad enough when the hardware drops messages.

## 3.0 The Meaning of a “Connection” In HIPPI-6400

(or, the name’s the same but the behavior isn’t necessarily identical....)

*(plan: incorporate a detailed explnation of the meaning of “scheduled” messaing on VC3 especially as compared with legacy HIPPI)*

## 4.0 Review of Physical Bandwidth Allocation and Some Usage Scenarios

(Note: eventually this would be the section in which modelling and simlation data would be reported)

## 5.0 Suggested Host Strategies

(Note: Guidelines for Hosts and Adapters to achieve whatever performance and behavior under uoad is desired would be added here after analysis and maybe modelling as well as reviewer suggestions.)

## 6.0 Comments on Switches

## 7.0 Comments on SNMP for Mixed HIPPI-6400/1600/800 Networks